p-ISSN : 2808-8751 (print)
e-ISSN : 2798-2513 (online)
http://journal.stiestekom.ac.id/index.php/TEKNIK

Deteksi Anomali Menggunakan Extended Isolation Forest (Eif)

Milka Wijayanti Sunarto[1], Dendy Kurniawan[2], Edy Siswanto[3], Haris Ihsanil Huda[4]
[1] Universitas Saind dan Teknologi Komputer Semarang, e-mail: milka@gmail.com
[2]Universitas Saind dan Teknologi Komputer Semarang, e-mail: dendy@stekom.ac.id
[3]Universitas Saind dan Teknologi Komputer Semarang, e-mail: edy@stekom.ac.id
[4]Universitas Saind dan Teknologi Komputer Semarang, e-mail: haris@stekom.ac.id

ARTICLE INFO

Article history:

Received 30 Agustus 2021
Received in revised form 2 September 2021
Accepted 10 September 2021
Available online 22 September 2021

ABSTRACT

Tujuan Utama: Tujuan dari penelitian ini adalah untuk mengembangkan algoritma deteksi anomali yang lebih efektif dan akurat menggunakan Extended Isolation Forest (EIF) dan mengimplementasikannya ke dalam platform sumber terbuka Machine Learning (ML) H2O-3. Background problem: Algoritma Isolation Forest (IF) asli menghadirkan bentuk deteksi baru, meskipun algoritme mengalami bias yang berasal dari percabangan pohon. Perpanjangan algoritme menghilangkan bias dengan menyesuaikan percabangan, dan algoritme asli hanya menjadi kasus khusus. EIF diimplementasikan ke dalam platform sumber terbuka ML H2O-3. kebaharuan: Kebaruan dari penelitian ini adalah penggunaan algoritma EIF dalam deteksi anomali. Selain itu, penelitian ini juga mengimplementasikan EIF ke dalam platform sumber terbuka ML H2O-3 untuk dijalankan pada sistem komputasi terdistribusi dengan pustaka Map/Reduce. Research Method: Penelitian ini menggunakan metode deteksi anomali dengan fokus pada algoritma EIF. temuan: Hasil pengujian menunjukkan bahwa Extended Isolation Model perlu disesuaikan. Tes kinerja deteksi anomali mengungkapkan sedikit ketidaksempurnaan dalam deteksi struktur data jika dibandingkan dengan satu-satunya implementasi algoritma Python yang tersedia. Hasil ujian untuk tahap evaluasi dinyatakan lulus dan waktu komputasi secara logaritmik lebih kecil dengan jumlah utas. Kesimpulan: pada penelitian selanjutnya, algoritma dapat ditingkatkan lebih lanjut dengan menskalakan anomali deteksi untuk data dimensi tinggi. Ini dapat diimplementasikan dengan menambahkan parameter lain yang memungkinkan metode pemilihan fitur dalam perhitungan..

Kata kunci: *Deteksi Anomali, Machine Learning, Komputasi terdistribusi.*

Received Agustus 30, 2021; Revised September 2, 2021; Accepted September 22, 2021

p-ISSN : 2808-8751 e-ISSN : 2798-2513

1. PENDAHULUAN

Dalam era pemrosesan dan penyimpanan informasi yang semakin meningkat dengan cepat, dikenal sebagai big data, tidak hanya diperlukan sarana, tetapi juga metode yang efektif untuk menganalisis dan mengevaluasi informasi tersebut. Hal ini menjadi sangat penting di sektor bisnis, di mana analisis yang akurat dapat memberikan keuntungan kompetitif yang signifikan dalam lingkungan yang sangat kompetitif, serta dalam bidang-bidang seperti akademik, perawatan kesehatan, ilmu komputer, dan bahkan penelitian luar angkasa. Semua bidang ini membutuhkan kemampuan yang mendesak dalam menganalisis data dalam jumlah besar untuk menggerakkan kemajuan mereka. Ada beberapa cara yang dapat digunakan untuk menganalisis data, salah satunya adalah metode yang dikenal sebagai deteksi anomali atau deteksi outlier. Dalam openelitian ini, hanya istilah deteksi anomali yang akan digunakan. Deteksi anomali membagi data menjadi dua jenis, yaitu pengamatan normal (nominal) yang merujuk pada data yang tidak memiliki karakteristik yang tidak biasa, seperti sampel, contoh, atau distribusi data; dan anomali (outlier) yang merupakan titik data yang secara positif atau negatif berbeda dari pengamatan normal. Anomali negatif dalam perawatan kesehatan dapat menandakan adanya penyakit, sementara dalam transaksi kartu kredit dapat menunjukkan adanya penipuan. Di bidang luar angkasa, anomali bisa menjadi objek baru yang belum diketahui, sedangkan dalam bisnis, dapat menjadi pelanggan baru yang penting.

Awal perkembangan deteksi anomali dapat ditelusuri kembali ke statistik pada abad ke-19. Sejak itu, deteksi anomali telah menjadi penting dalam berbagai domain dan telah berkembang secara terpisah di setiap domain tersebut. Oleh karena itu, banyak teknik deteksi anomali yang menggunakan karakteristik data yang spesifik untuk bidang atau industri tertentu. Sebelum era big data, belum ada upaya untuk mengembangkan teknik deteksi anomali yang bersifat umum. Namun, seiring dengan evolusi analisis data, para peneliti mulai mempelajari teknik-teknik spesifik, mendefinisikan karakteristik data, dan mencoba mengembangkan algoritme deteksi anomali yang dapat diterapkan secara umum dan dapat diskalakan di berbagai domain. Tujuan dari penelitian ini adalah untuk mengimplementasikan algoritma EIF di Java. Algoritme akan diimplementasikan ke dalam platform sumber terbuka ML H2O-3 yang dikembangkan oleh perusahaan H2O.AI. H2O-3 adalah salah satu produk perusahaan untuk ML terdistribusi, yang ditulis dalam Java, menggunakan kerangka Peta/Pengurangan untuk komputasi awan. Oleh karena itu implementasi EIF di Java akan diuji, dan hasilnya dibandingkan dengan satu-satunya implementasi yang diketahui di Python. Selanjutnya, implementasi algoritma Java akan dibandingkan dengan implementasi open-source IF yang ada di berbagai library, termasuk H2O.AI.

2. TINJAUAN PUSTAKA

Deteksi Anomali

Pada awalnya, perlu ditentukan mengapa anomali harus dideteksi dan apa yang harus dilakukan dengannya. “Anomali harus diidentifikasi untuk studi lebih lanjut karena pengetahuan ini dapat digunakan untuk membuat model yang lebih kuat yang dapat memperhitungkan kemungkinan adanya anomali bahkan tanpa tertarik pada anomali spesifik” (Zimek et al., (2018). Kebaruan dapat dianggap sama dengan anomali, perbedaannya adalah, katakanlah, nilai bisnis dari titik yang terdeteksi. Anomali, pada dasarnya, adalah titik menarik yang berbeda dari titik normal tanpa ada peluang untuk menjadi titik normal. Kebaruan, di sisi lain, adalah titik (dalam banyak kasus beberapa titik), pertama kali terdeteksi sebagai anomali, tetapi setelah intervensi pakar domain, titik tersebut menjadi pengamatan normal. Dari sudut pandang algoritma deteksi anomali, kebaruan dan anomali memiliki karakteristik yang sama, tetapi kebaruan kemudian dimasukkan dalam distribusi normal jika model dibangun kembali. Chalapathy et al., (2019) and Zimek et al., (2018) mengatakan bahwa dalam hal implementasi, kebaruan memiliki ambang batas yang lebih rendah daripada ambang anomali. Alasan mengapa deteksi anomali diperlukan cukup jelas. Namun ada beberapa sudut pandang yang berbeda untuk deteksi anomali atau anomali itu sendiri. Ini adalah sisa-sisa penelitian terpisah sebelumnya dalam satu domain tanpa fokus pada generalisasi. Salah satu jenis anomali yang digunakan dalam penelitian ini adalah Point Anomalies. Instance sangat berbeda dari data lainnya dengan anomali yang sederhana dan intuitif (Chalapathy et al., (2019).

Data untuk Deteksi Anomali

Deteksi anomali menangani semua jenis data. Gambar, deret waktu, atau data tabular sangat umum. Untuk keperluan penelitian ini hanya data dalam struktur tabular, yang berarti baris dan kolom, yang akan dipertimbangkan. Pemrosesan awal data terserah pengguna. Algoritma berharap bahwa tipe data adalah bilangan real. Baris juga direferensikan sebagai variabel x atau titik, observasi, instance, dan kolom

p-ISSN : 2808-8751 e-ISSN : 2798-2513

dapat dialamatkan sebagai fitur. Dimungkinkan untuk menandai baris sebagai normal atau sebagai anomali. Tanda ini disebut label. Variabel ”N” adalah singkatan dari jumlah baris dan variabel ”P” adalah jumlah kolom (Fitur). Data tersebut digunakan untuk membuat model deteksi anomali. Pembuatan model disebut juga building, training, atau learning. Model yang dibuat kemudian digunakan untuk menentukan apakah instance tersebut merupakan anomali atau bukan. Proses ini dapat dirujuk sebagai evaluasi, pengujian, atau prediksi.

Output Deteksi Anomali

Perbedaan antara algoritma yang digunakan untuk deteksi anomali, selain teknik yang digunakan, adalah outputnya. Terdapat teknik yang memberikan klasifikasi dua nilai, titik normal, dan anomali; biasanya ini adalah algoritma yang berasal dari keluarga pengelompokan (2). Output kedua yang mungkin adalah skor anomali dan pengguna harus menentukan ambang untuk anomali. Skor kemungkinan besar merupakan variabel kontinu antara 0 dan 1 atau antara -1 dan 0 (tergantung implementasinya). Penelitian ini selalu mempertimbangkan rentang [0, 1] di mana skor anomali mendekati nol berarti titik normal dan skor mendekati 1 berarti anomali. Memang, kemiripan dengan tipologi ilmu data umum tentang regresi dan klasifikasi sudah jelas. Perbedaannya di sini adalah keluaran harus selalu diberi label sebagai normal atau sebagai anomali. Dalam kasus pertama, label langsung berasal dari output. Dalam kasus kedua ada kesempatan untuk mempelajari distribusi skor anomali, dengan pengguna memutuskan apa yang anomali dan apa yang bukan anomali.

Jenis Teknik Deteksi Anomali

Studi oleh Chandola et al., (2009) dan Chalapathy et al., (2019) memberikan taksonomi teknik deteksi anomali, pertama mengeksplorasi teknik data mining tradisional sambil juga mencakup Deep Learning (DL), dimana ini merupakan metode berdasarkan Artificial Neural Networks (ANN) buatan, studi kedua hanya berfokus pada DL. Kedua penelitian tersebut mengurutkan teknik deteksi dari dua sudut pandang. Pertama data yang diberikan; dan kedua teknik algoritma. Perhatikan bahwa penyortiran tidak eksklusif. Teknik dapat berbeda di antara jenisnya, tetapi tujuannya adalah untuk menentukan asumsi algoritme.

Input

Tipologi sederhana dan paling umum dari setiap teknik data mining adalah dengan input yang diberikan ke algoritma tak terkecuali deteksi anomali. Teknik-teknik tersebut dapat dibagi menjadi tiga kelas untuk data mining dan DL. Chalapathy et al., (2019) juga menyediakan dua kelas lagi berdasarkan kombinasi DL dan data mining. Misalkan data dengan hanya dua label, normal dan anomali, pertama adalah *“S*upervised*”* dimana kedua label disediakan, tetapi datanya sangat tidak seimbang sehingga terdapat pengamatan normal yang tak terhitung jumlahnya dan beberapa anomali. Jenis data ini jarang terjadi, dan tekniknya sama dengan membangun model prediktif dengan data yang sangat tidak seimbang. Kedua adalah *“*Semi-Supervised*”* dimana hanya satu label yang disediakan, kemungkinan besar untuk observasi normal, tetapi memiliki label hanya untuk anomali juga dimungkinkan dalam beberapa kasus. Teknik yang beroperasi di kelas ini kemungkinan besar membangun model untuk observasi normal dan menggunakannya untuk mengidentifikasi anomali. Terakhir adalah *“*Unsupervised*”* dimana idak ada label yang disediakan. Jenis deteksi anomali ini adalah yang paling luas. Asumsi penting untuk deteksi anomali adalah bahwa pengamatan normal jauh lebih sering daripada anomali. Teknik tanpa asumsi ini disebut pengelompokan biner.

Pertimbangkan sifat deteksi anomali

Hanya tipe algoritma Semi-Supervised dan Unsupervised yang diperhitungkan. Teknik yang diawasi dikecualikan dari ruang lingkup karena ini adalah data mining dengan data yang sangat tidak seimbang. IF (Diperluas) milik bagian Tanpa Pengawasan. Dua teknik tambahan yang diusulkan oleh Chalapathy et al., (2019) adalah: *Deteksi anomali dalam hibrid.* Model DL digunakan sebagai ekstraktor fitur untuk mempelajari fitur yang kuat. Fitur-fiturnya adalah input ke dalam model ML apa pun. Algoritme yang didasarkan pada deteksi hibrid dapat diskalakan ke data berdimensi tinggi. Kedua, *Jaringan Syaraf Satu Kelas*. Jaringan Syaraf Satu kelas menggabungkan kemampuan jaringan dalam untuk mengekstrak representasi data yang semakin kaya bersama dengan tujuan satu kelas. Model tersebut secara bersamaan melatih jaringan saraf yang dalam, dan mengoptimalkan penyertaan data. Anomali tidak mengandung

p-ISSN : 2808-8751 e-ISSN : 2798-2513

faktor umum sehingga penutupan data gagal pada mereka. Keuntungan yang merugikan adalah waktu yang lama diperlukan untuk melatih model.

3. Algoritma Deteksi Anomali

K-Means

Algoritma yang merupakan penyesuaian dari keluarga berbasis cluster adalah K-means, yang merupakan salah satu algoritma pengelompokan yang sangat terkenal. K-means melakukan pembentukan kluster dari titik-titik data yang serupa, dengan jumlah kluster yang ditentukan sebelumnya ("K"). Algoritma ini bergantung pada asumsi penting bahwa instance-data normal akan termasuk dalam salah satu dari "K" grup yang terbentuk, sementara titik-titik yang berada di luar grup dapat dianggap sebagai potensi anomali. Dalam implementasi K-means, titik-titik yang disebut centroid dihitung untuk menentukan pusat dari setiap kluster "K", dan titik-titik yang lebih dekat ke salah satu centroid akan diberi label sebagai anggota dari kluster tertentu. Algoritma terkenal dapat dengan mudah disesuaikan untuk deteksi anomali dengan menambahkan ambang batas untuk jarak antara pusat pengamatan normal dan titik yang dianalisis. Terdapat beberapa metode yang berbeda untuk mengklasifikasikan anomali, tetapi semuanya menggunakan semacam jarak yang disebutkan ke pusat massa. Output dari algoritma ini adalah klasifikasi titik sebagai normal atau sebagai anomali (Li et al., (2007); DataScience.com (2020); Ahmed et al., (2016). Keuntungan signifikan dari pendekatan ini adalah sederhana dan deteksi anomali mudah dipahami. Meskipun demikian, algoritme ini memiliki kelebihan, yang harus dipertimbangkan secara kritis sebelum memutuskan untuk menggunakan algoritme dalam penerapan produksi. Pertama, pemilihan hyperparameter ”K” (Ahmed et al., (2016). Ada teknik untuk memilih K, tetapi sulit untuk menetapkan sebuah jumlah cluster. Hal ini terutama terjadi pada data berdimensi tinggi, di mana kemungkinan visualisasi terbatas. Masalah kedua dari K-means adalah hanya cocok ketika cluster diharapkan memiliki bentuk yang relatif teratur (Gumbao et al., (2020). Ketiga, anomali itu sendiri memengaruhi perhitungan pusat massa. Ini seharusnya tidak menjadi masalah yang signifikan karena anomali jarang terjadi, tetapi ini merupakan kerugian dibandingkan dengan algoritma lainnya (Ahmed et al., (2016); (Gumbao et al., (2020). Keempat, karena tidak ada batas bawah untuk jumlah titik dalam sebuah klaster, dimungkinkan untuk membuat klaster anomali yang terdefinisi sempurna dengan "K" yang tidak diketahui (Gumbao et al., (2020).

DBSCAN

Algoritma yang disajikan oleh Schubert et al., (2017) mendefinisikan terminologi untuk titik-titik dalam data. Algoritma dimulai dengan titik acak, titik dianggap berada di cluster yang sama ketika memiliki nilai lebih tinggi dari "minPts" tetangganya dalam radius ε (termasuk titik kueri). Titik anomali adalah titik dengan nilai kurang dari tetangga "minPts" dalam radius ε. Proses ini berlanjut untuk setiap titik sampai tidak ada lagi titik yang belum diputuskan. Perlu ditambahkan bahwa algoritma DBSCAN awalnya disajikan pada tahun 1996 oleh Ester et al., (2020) dan menjadi ide inti untuk variasi masa depan seperti ***HDBSCAN****, OPTIK atau LSDBC. Hal yang sama terjadi dengan algoritma IF - khususnya, DeepIF (Li et al., (2020), iForestASD (Ding et al., (2013), EIF (Hariri et al., (2019) dan beberapa lainnya. DBSCAN cocok untuk memisahkan klaster dengan kepadatan tinggi dari klaster dengan kepadatan rendah di sisi lain, algoritme berjuang dengan kelompok dengan kepadatan yang berbeda-beda. Meskipun DBSCAN memisahkan data dengan klaster yang berubah bentuk, data tersebut menderita jika data memiliki terlalu banyak dimensi. Selain itu, output algoritme dapat bervariasi tergantung pada apakah dimulai dengan pilihan acak dari titik pertama seperti K-means, tetapi jumlah anomali tidak berpengaruh pada penghitungan titik klaster. Tujuan DBSCAN adalah untuk secara tepat memperkirakan kelompok titik normal sambil mempertimbangkan kemungkinan terjadinya titik anomali. Namun, menurut Chandola et al., (2009) DBSCAN tidak dioptimalkan untuk menemukan anomali (Lutins, (2020).

Mendukung Ekstensi Mesin Vektor Untuk Deteksi Anomali

SVM adalah teknik kernel yang biasanya dikaitkan dengan pembelajaran terawasi; namun, ada penyesuaian dari algoritme asli untuk pembelajaran tanpa pengawasan yang cocok untuk deteksi anomali (DataScience.com (***2020***). Koneksi teknik berbasis kernel dan pendekatan berbasis kepadatan diringkas dengan baik dalam kutipan berikut: “Algoritma pengelompokan adalah contoh lebih lanjut dari teknik pembelajaran tanpa pengawasan yang dapat dikernel. Sudut pandang ekstrim adalah bahwa pembelajaran tanpa pengawasan adalah tentang memperkirakan kepadatan. Jelas, pengetahuan tentang kerapatan P kemudian akan memungkinkan kita untuk memecahkan masalah apa pun yang dapat diselesaikan berdasarkan data tersebut.” (Zhang et al., (2006). Algoritma Zhang et al., (2006) menghitung fungsi biner, yang diharapkan untuk menangkap daerah dalam ruang input dimana kerapatan probabilitas titik

p-ISSN : 2808-8751 e-ISSN : 2798-2513

normal. Secara khusus, fungsi sedemikian rupa sehingga sebagian besar data akan berada di wilayah di mana fungsinya bukan nol. Tujuannya adalah untuk mengikat probabilitas bahwa titik anomali dari distribusi yang sama akan berada di luar wilayah yang diestimasi dengan margin tertentu.

Autoencoder

Autoencoder atau juga Replicator Neural Networks (RNN) mewakili data di dalam beberapa lapisan tersembunyi dengan rekonstruksi iteratif dari data input. Autoencoder yang terlatih dengan baik dapat dengan tepat merekonstruksi contoh normal apa pun, tetapi untuk anomali, mereka menghasilkan kesalahan yang signifikan selama rekonstruksi. Poin yang menghasilkan jumlah kesalahan yang tinggi dianggap sebagai anomali (Chalapathy et al., (2019). Pilihan arsitektur autoencoder tergantung pada sifat data, jaringan konvolusi lebih disukai untuk gambar sedangkan model berbasis memori jangka pendek (LSTM) cenderung menghasilkan hasil yang baik untuk data sekuensial." (Chalapathy et al., (2019). Autoencoder adalah NN yang sederhana dan efektif untuk deteksi anomali, tetapi kinerjanya menurun jika autoencoder berurusan dengan data pelatihan yang mengandung banyak noise (Chalapathy et al., (2019). Zhou et al., (2017) memberikan cara untuk membuat Robust Deep Autoencoder (RDA) yang terinspirasi oleh Robust PCA. Algoritme menganggap bahwa data kata asli berisik dan berurusan dengan ini melalui metode yang didasarkan pada pemisahan data masukan menjadi dua kelas. Satu kelas berisi data yang dapat dengan mudah direproduksi, dan kelas kedua berisi anomali dan derau. Setelah pemisahan, autoencoder dilatih pada data yang bersih dan bebas noise dan dengan demikian memberikan perkiraan yang kuat dari pengamatan normal. Rincian metode itu sendiri ada di kertas (Zhou et al., (2017). Penulis membandingkan Robust Deep Autoencoder dengan implementasi Scikit-learn dari IF (Scikit-learn (2020) pada versi berisik dari data MNIST dan menyimpulkan bahwa RDA meningkatkan skor F1 sekitar 73%. Alasan mengapa performa IF lebih buruk mungkin karena gambar 28x28 piksel, yang menghasilkan 784 fitur. IF dapat bekerja lebih baik bila dikombinasikan dengan metode ekstraksi fitur.

Isolation Forest (IF)

Algorithm IF mengusulkan teknik baru pendeteksian anomali yang berbeda dari yang sebelumnya. Algoritma dibangun berdasarkan asumsi bahwa anomali adalah "sedikit dan berbeda". Alih-alih membuat profil distribusi normal dan mengukur jarak setiap titik ke titik normal, algoritme malah mengisolasi anomali dari titik lainnya dan mengukur perbedaannya menggunakan Pohon Isolasi (iTree). Pendekatan ini, bersama dengan ansambel Isolation Trees dan metode sub-sampling, menghasilkan model dengan kebutuhan memori yang rendah serta kompleksitas logaritmik yang rendah dari tahapan pelatihan dan evaluasi (Liu et al., (2008).

Konsep Isolasi

Dalam konteks ini, istilah isolasi berarti "memisahkan sebuah contoh dari sisa contoh". Misalkan Pohon Keputusan Biner yang diinduksi data dengan hadiah anomali. Asumsi "sedikit dan berbeda" menyiratkan anomali diputuskan lebih dekat ke akar, dan titik normal lebih dalam di pohon. Pohon biner dibangun untuk mengisolasi semua titik dan mengukur Panjang Jalur masing-masing dari akarnya. Definisi berikut berasal dari kertas (Liu et al., (2008).

Komputasi skor anomali

Output dari algoritma IF adalah skor anomali. Mempertimbangkan ambang batas yang ditentukan pengguna, algoritme IF memberi label pada masing-masing titik sebagai normal atau sebagai anomali. Singkatnya, skor anomali adalah rata-rata h(x) di iF orest yang dinormalisasi oleh jalur rata-rata pencarian yang gagal di Binary Search Tree (BST

Algoritma Isolasi Hutan

Hyperparameter dari model IF adalah t (jumlah pohon), ψ (ukuran sub-sampling). Algoritma dibagi menjadi dua tahap. Pertama adalah tahap Pelatihan dimana hutan tercipta. Tahap kedua adalah tahap Evaluasi, yang menempatkan titik tertentu ke dalam setiap pohon dan memberikan rata-rata h(x) dari titik tersebut.

Pada tahap pelatihan, sub-sampling dilakukan dan ansambel rees TI dibangun. Tinggi setiap pohon iT dibatasi oleh langit-langitnya ($\log 2\ \psi$), yang kira-kira merupakan tinggi rata-rata BST untuk ukuran data yang diberikan (Liu et al., (2008). Setiap iTree berfokus pada titik-titik yang memiliki jalur lebih pendek dari rata-rata. Algoritma untuk pelatihan dipisahkan menjadi dua fungsi. Rekursi digunakan dalam

p-ISSN : 2808-8751 e-ISSN : 2798-2513

Algoritma 2 untuk membangun iTree. Output dari tahap pelatihan adalah iF orest yang disiapkan untuk penilaian setiap poin yang diberikan.

Algorithm 1 $iForest(X, t, \psi)$

Require: X - input data, t - number of trees, ψ - sub-sampling size
Ensure: a set of t $iTrees$

Initialize $Forests$
set height limit $l = ceiling(\log_2 \psi)$
for $i = 1$ to t **do**
    $X' \leftarrow sample(X, \psi)$
    $Forest \leftarrow Forest \cup iTree(X', 0, l)$
end for
return $Forest$

Algorithm 2 $iTree(X, e, l)$

Require: X - input data, e - current tree height, l - height limit
Ensure: an iTree

if $e \geq l$ or $|X| \leq 1$ **then**
    return $exNode\{Size \leftarrow |X|\}$
else
    let Q be a list of attributes in X
    randomly select an attribute $q \in Q$
    randomly select a split point p from max and min $values$ of attribute q in X
    $X_l \leftarrow filter(X, q \leq p)$
    $X_l \leftarrow filter(X, q > p)$
    return inNode$\{Left \leftarrow iTree(X_l, e+1, l),$
        $Right \leftarrow iTree(X_r, e+1, l),$
        $SplitAtt \leftarrow q,$
        $SplitValue \leftarrow p\}$
end if

Pada tahap evaluasi, algoritma keluaran dari tahap evaluasi adalah h(x) dari titik yang diberikan. Rata-rata h(x) dalam iF orest dihitung dan diserahkan ke rumus skor anomali. Untuk solusi untuk dimensi tinggi, Algoritma IF mengalami kutukan dimensi. Liu et al., (2008) merancang solusi berdasarkan penggabungan algoritma IF dengan metode seleksi fitur. Sub-set fitur berdasarkan uji statistik Kurtosis dipilih dari sub-sampel data sebelum setiap iTree dibangun. Eksperimen Liu et al., (2008) menunjukkan keunggulan waktu IF karena persyaratan pemrosesan yang rendah pada data dimensi tinggi.

Keuntungan dan Kerugian Isolasi

Algoritma IF menggunakan perspektif yang berbeda untuk mendeteksi anomali. Daripada membuang-buang sumber daya untuk menentukan distribusi normal secara tepat, ini justru berfokus pada isolasi anomali. Evaluasi empiris Liu et al., (2008) menunjukkan bahwa IF bekerja secara signifikan lebih baik daripada metode berbasis jarak kompleksitas waktu yang hampir linier. Namun, solusi berbasis DL memiliki kinerja yang lebih baik daripada IF (Chalapathy et al., (2019). Berbeda dengan penyesuaian K-means yang harus memiliki hyperparameter "K" yang ditentukan, tidak perlu memasukkan jumlah cluster di IF. Hal ini menyebabkan kurangnya pemeliharaan model produksi. Jika data produksi berkembang, ada kemungkinan untuk menetapkan ambang batas untuk hal baru seperti yang disarankan (Chalapathy et al., (2019). Di sisi lain, tidak ada kemungkinan untuk memperbarui model secara terus-menerus - karenanya jika data berkembang secara signifikan, model harus dibangun kembali. Selain itu, algoritme IF tidak memiliki keunggulan minPts dari algoritme DBSCAN, yang memungkinkan pembuatan batas bawah untuk sejumlah titik yang diperlukan untuk membentuk wilayah padat. Namun demikian IF dioptimalkan untuk menemukan anomali, bukan untuk mendefinisikan distribusi normal. Distribusi skor anomali juga harus disebutkan. Skor anomali mengalami bias yang disebabkan oleh cara pembagian iT. Hariri et al., (2019) mengusulkan solusi sederhana namun cerdik untuk menghilangkan bias ini dan membangun model yang lebih kuat dengan kinerja yang sama. Oleh karena itu dapat disimpulkan bahwa IF adalah kasus khusus dari metode isolasi umum.

p-ISSN : 2808-8751 e-ISSN : 2798-2513

Extended Isolation Forest (EIF)

EIF adalah algoritma untuk deteksi anomali tanpa pengawasan berdasarkan algoritma IF. Ekstensi terletak pada generalisasi metode percabangan Pohon Isolasi. Percabangan IF asli menyediakan pemotongan hanya sejajar dengan salah satu sumbu. Metode percabangan EIF memungkinkan pemotongan data menggunakan hyperplane dengan kemiringan acak. Motivasi algoritma EIF adalah studi skor anomali yang diberikan oleh IF pada data mainan 2-D. Kemunculan kelompok hantu terbukti. Hal ini dapat menyebabkan deteksi anomali positif palsu karena ambang batas digunakan untuk menemukan (Hariri et al., (2019). Jika pemisahan setiap titik divisualisasikan dan dihubungkan dengan score map, maka dapat disimpulkan bahwa hal itu disebabkan oleh metode pemisahan.Cara percabangan BST membuat garis sejajar dengan salah satu sumbu, yang pada gilirannya menimbulkan bias (Hariri et al., (2019).

Generalisasi IF

Visualisasi pemisahan BST mengarah pada gagasan bahwa percabangan harus lebih acak dan tidak hanya terfokus pada garis yang sejajar dengan sumbu. Hariri et al., (2019) mengusulkan dua metode untuk generalisasi. Pertama cocok untuk solusi yang tidak memungkinkan untuk mengubah algoritme IF. Ini mengusulkan rotasi sub-sampel sebelum setiap iTree dibangun, sehingga "merata-ratakan" bias. Metode kedua memperkenalkan penyesuaian untuk algoritma IF yang mengarah ke generalisasi lengkap dari metode IF, yang disebut EIF.

Pohon Berputar

Original IF membuat sub-sampel data sebelum setiap iTree dibangun. Bias dapat dirata-ratakan jika setiap iTree dibangun di atas sub-sampel yang diputar. Dalam kasus seperti itu, bias tetap ada, namun berbeda untuk setiap iTree dan bias tersebut dirata-ratakan. Meskipun ini mengarah ke hasil yang lebih baik, metode ini masih belum cukup karena Setiap pohon IT masih mengalami bias (walaupun bias berbeda untuk setiap pohon), informasi tambahan tentang sudut rotasi untuk setiap pohon perlu disimpan, jika data kurang simetris, metode ini tidak cukup dan tTidak cocok untuk data berdimensi besar dan tinggi.

Penyesuaian cabang

Penyebab bias adalah bahwa percabangan ditentukan oleh kesamaan dengan BST. Pada setiap titik percabangan fitur dan nilainya dipilih; ini memperkenalkan bias karena titik percabangan sejajar dengan salah satu sumbu. Kasus umum perlu menentukan kemiringan acak untuk setiap titik percabangan. Alih-alih memilih fitur dan nilai, ia memilih kemiringan acak ~n untuk potongan percabangan dan perpotongan acak ~p. Kemiringan dapat dihasilkan dari N (0, 1) distribusi Gaussian, dan intersep dihasilkan dari distribusi seragam dengan batas-batas yang berasal dari sub-sampel data yang akan dipisah (Hariri et al., (2019). Pada titik ini, hyperparameter generalisasi baru, extensionLevel, diperkenalkan. Fungsi extensionLevel adalah untuk memaksa item acak ~n menjadi nol. Nilai hyperparameter extensionLevel adalah antara 0 dan (P − 1). Nilai 0 berarti semua lereng akan sejajar dengan semua sumbu, yang sesuai dengan perilaku IF. Semakin tinggi jumlah extensionLevel menunjukkan bahwa pemisahan akan sejajar dengan extensionLevel-jumlah sumbu. Ekstensi penuh berarti Tingkat ekstensi sama dengan P − 1. Hal ini menunjukkan bahwa kemiringan titik percabangan akan selalu diacak. Hariri et al., (2019) merekomendasikan untuk menggunakan EIF yang diperluas sepenuhnya. Ekstensi yang lebih rendah cocok untuk domain di mana rentang minimum dan maksimum untuk setiap fitur sangat berbeda (Hariri et al., (2019).

Algoritma EIF

Ini adalah penyajian pseudo-code of Branching adjustment oleh Hariri et al., (2019). Metode pertama dari Rotated Trees (iTree) tidak termasuk dalam pseudo-code atau implementasi selanjutnya. Algoritma juga dipisahkan menjadi dua tahap. Pertama untuk bangunan iForest dan kedua untuk perhitungan h(x). Hyperparameter dari model EIF adalah t sebagai jumlah pohon, dan ψ sebagai ukuran sub-sampling, extensionLevel dengan nilai dalam range [0, P − 1]; di mana P adalah jumlah fitur.

Dalam tahap pelatihan bagian pertama dari algoritma diketahui dari IF asli. Tidak ada perbedaan dalam kode semu. Perubahannya terletak pada Algoritma 5, tepatnya pada baris 4-8, dimana percabangannya disesuaikan. Baris 9 menunjukkan bahwa iTree berisi vektor dan tidak ada nilai tunggal. Ukuran vektor adalah jumlah fitur. Dengan demikian kebutuhan memori tidak meningkat secara signifikan karena jumlah fitur tetap jauh lebih rendah daripada jumlah baris.

p-ISSN : 2808-8751 e-ISSN : 2798-2513

Algorithm 4 $iForest(X, t, \psi, extensionLevel)$

Require: X - input data, t - number of trees, ψ - sub-sampling size
Ensure: a set of t $iTrees$
1: **Initialize** $Forest$
2: set height limit $l = ceiling(\log_2 \psi)$
3: **for** $i = 1$ to t **do**
4: $X' \leftarrow sample(X, \psi)$
5: $Forest \leftarrow Forest \cup iTree(X', 0, l, extensionLevel)$
6: **end for**
7: **return** $Forest$

Algorithm 5 $iTree(X, e, l, extensionLevel)$

Require: X - input data, e - current tree height, l - height limit
Ensure: an iTree
1: **if** $e \geq l$ or $|X| \leq 1$ **then**
2: **return** $exNode\{Size \leftarrow |X|\}$
3: **else**
4: randomly select a normal vector $\vec{n} \in IR^{|X|}$ by drawing each coordinate of $\vec{n}$ from a standard Gaussian distribution.
5: randomly select an intercept point $\vec{p} \in IR^{|X|}$ in the range of X
6: set coordinates of $\vec{n}$ to zero according to extension level
7: $X_l \leftarrow filter(X, (X - \vec{p}) \cdot \vec{n} \leq 0)$
8: $X_r \leftarrow filter(X, (X - \vec{p}) \cdot \vec{n} > 0)$
9: **return** inNode$\{Left \leftarrow iTree(X_l, e + 1, l)$,
$Right \leftarrow iTree(X_r, e + 1, l)$,
$Normal \leftarrow \vec{n}$,
$Intercept \leftarrow \vec{p}\}$
10: **end if**

Tahap evaluasi

Tahap evaluasi disesuaikan dengan tahap pelatihan. Fokus pada baris 6 dan 8 pada Algoritma 6, di mana percabangan disesuaikan. Seperti pada IF, rata-rata h(x) dihitung dan dimasukkan ke dalam rumus skor anomali.

Algorithm 6 $PathLength(\vec{x}, T, e)$

Require: $\vec{x}$ - an instance, T - an iTree, e - current path length; to be initialized to zero when first called
Ensure: path length of $\vec{x}$
1: **if** T is an external node **then**
2: **return** $e + c(T.size)$ {$c(.)$ is defined in Equation (2.1)}
3: **end if**
4: $\vec{n} \leftarrow T.Normal$
5: $\vec{p} \leftarrow T.Intercept$
6: **if** $(\vec{x} - \vec{p}) \cdot \vec{n} \leq 0$ **then**
7: **return** $PathLength(\vec{x}, T.left, e + 1)$
8: **else if** $(\vec{x} - \vec{p}) \cdot \vec{n} > 0$ **then**
9: **return** $PathLength(\vec{x}, T.rigth, e + 1)$
10: **end if**

Perbandingan IF

EIF adalah generalisasi dari IF, EIF dapat sepenuhnya menggantikan IF karena memungkinkan untuk meningkatkan generalisasi dengan menggunakan hyperparameter extensionLevel. Nilai minimum hyperparameter adalah 0, yang sesuai dengan perilaku IF. Maksimumnya adalah P − 1 dan singkatan dari EIF yang diperpanjang penuh. Alasan mengapa IF masih harus diperhitungkan sebagai alternatif adalah kurangnya implementasi EIF. Menurut pengetahuan terbaik dari platform data mining, satu-satunya

p-ISSN : 2808-8751 e-ISSN : 2798-2513

implementasi publik saat ini adalah yang dilakukan oleh penulis (Hariri et al., (2019). Ini adalah implementasi Python yang tidak didukung oleh platform ML mana pun, yang membuatnya tidak cocok untuk solusi siap produksi. Bahkan jika implementasi EIF sudah cukup, IF harus tetap digunakan karena kemampuan interpretasinya. Karena IF menggunakan BST di dalamnya, maka mudah untuk menginterpretasikan mengapa suatu titik dianggap sebagai anomali. Tentu saja, ansambel pohon juga menyulitkan, tetapi setidaknya ada peluang. Interpretabilitas EIF hilang dengan penyesuaian percabangan.

Implementasi Extended saat ini

IF memiliki terdapat kekurangan dalam implementasi EIF. Implementasi hanya dilakukan oleh para pendiri (Hariri et al., (2019). Alasan utama mengapa penerapannya tidak dapat digunakan dalam lingkungan produksi adalah kurang dukungan, selain itu paket Python tidak bekerja pada semua platform sesuai dengan masalah yang dibuka (PoradaKev (2020). Parameter benih yang hilang untuk membuat algoritme deterministik di beberapa proses, dan meskipun algoritme itu sendiri memiliki persyaratan sumber daya yang rendah, implementasi di Python tidak memungkinkan pemrosesan big data.

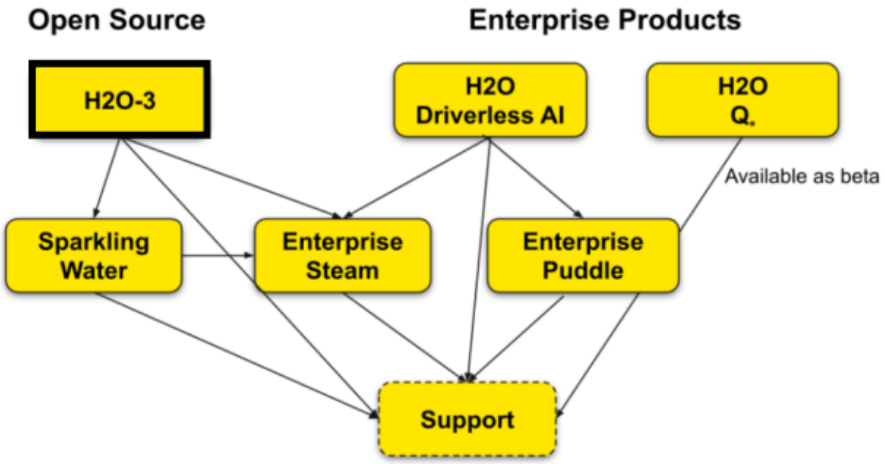
IMPLEMENTASI

Ada beberapa opsi cara membuat algoritme ML menjadi publik. Cara cepat dan mudah adalah mengimplementasikan paket Python dan menerbitkannya di PyPi (joelbarmettlerUZH, (2020), sebuah repositori dari semua paket Python publik. Dengan mengimplementasikan fungsi dalam paket Python, komunitas pengguna yang luas tercakup. Pilihan lainnya adalah membuat repositori publik di GitHub (GitHub, (2020) untuk komunikasi dengan pengguna dan/atau kerja sama dengan kontributor lain. Dalam kasus di mana implementasi Python tidak mencukupi (misalnya, karena alasan kinerja), Java dapat digunakan sebagai gantinya. Mengikuti implementasi algoritma dapat dipublikasikan di repositori Maven (Foundation, (2020). Dari sudut pandang pengguna, Java sendiri umumnya tidak dikenal untuk ML, dan penerbitannya akan memakan waktu lebih lama karena penerapannya perlu melalui proses persetujuan. Pilihan terbaik untuk membuat algoritme dapat digunakan dan ramah pengguna adalah dengan mengkontribusikan penerapannya ke dalam platform sumber terbuka. Beberapa dari formulir platform ini bahkan menawarkan dukungan pengguna berbayar, yang meningkatkan tingkat perbaikan masalah atau bug pengguna. Ini pada gilirannya mengarah pada implementasi yang lebih baik dari waktu ke waktu. Salah satu platform tersebut adalah pustaka Scikit-learn berbasis Python. Namun platform ini selektif ketika menambahkan algoritma baru karena sumber daya yang terbatas untuk pemeliharaan kode (Scikit-learn, (2020). Roy, (2019) adalah daftar platform ML lainnya dengan fokus yang luas. Beberapa dari mereka fokus pada komputasi GPU (TensorFlow, (2020), Neural Networks yang dapat diskalakan (Caffe, (2020), operasi aljabar linier yang dapat diskalakan (MAHOUT, (2020) dan banyak lagi. Antarmuka Python adalah persyaratan "harus dimiliki" untuk semua platform yang ingin digunakan secara luas. Namun, kunci untuk pengembang adalah mesin komputasi, terkadang ditulis dengan Python itu sendiri, tetapi C++, Java, atau Scala juga merupakan alternatif yang memungkinkan. Meskipun mulai berkontribusi ke platform sumber terbuka cukup mudah, mengimplementasikan algoritme baru dan menerima persetujuan seringkali merupakan proses yang panjang, terlepas dari ukuran atau kerumitan implementasinya. Penelitian ini merupakan hasil kerjasama dengan perusahaan H2O.AI, yang setuju untuk mengimplementasikan EIF ke dalam H2O-3 library mereka sebagai penawaran pelengkap dari IF asli yang telah diimplementasikan.

H2O.AI

H2O.AI adalah perusahaan perangkat lunak sumber terbuka yang menawarkan berbagai produk (platform) dan layanan untuk ML. Produk membantu pengguna untuk membangun model dan menjalankannya pada sistem produksi. Fokus H2O.AI adalah menyediakan solusi ML yang dapat diskalakan. Produk dirancang dan disiapkan tidak hanya untuk memproses kumpulan data eksperimen kecil dalam satu file, tetapi produk yang sama juga dapat beroperasi dengan teknologi big data seperti Hadoop, Spark, dan memproses data terabyte tanpa campur tangan pengguna. Setelah pelatihan selesai, model dapat diserialisasi menjadi artefak siap produksi. Pada Gambar 2 adalah portofolio produk H2O.AI. Ada produk open-source dan perusahaan dalam portofolio, semuanya dengan kemungkinan dukungan berbayar (H2O.AI. H2O, (2020). Salah satu alat H2O.AI adalah teknologi AutoML. Ini adalah alat yang membantu ilmuwan data dengan tugas-tugas mereka seperti ekstraksi fitur, berjalan melalui algoritme ML dan hyperparameter mereka, menemukan model terbaik, dan terakhir, menyebarkan model langsung ke lingkungan produksi (H2O.AI. H2O, (2020); Matousek, (2020).

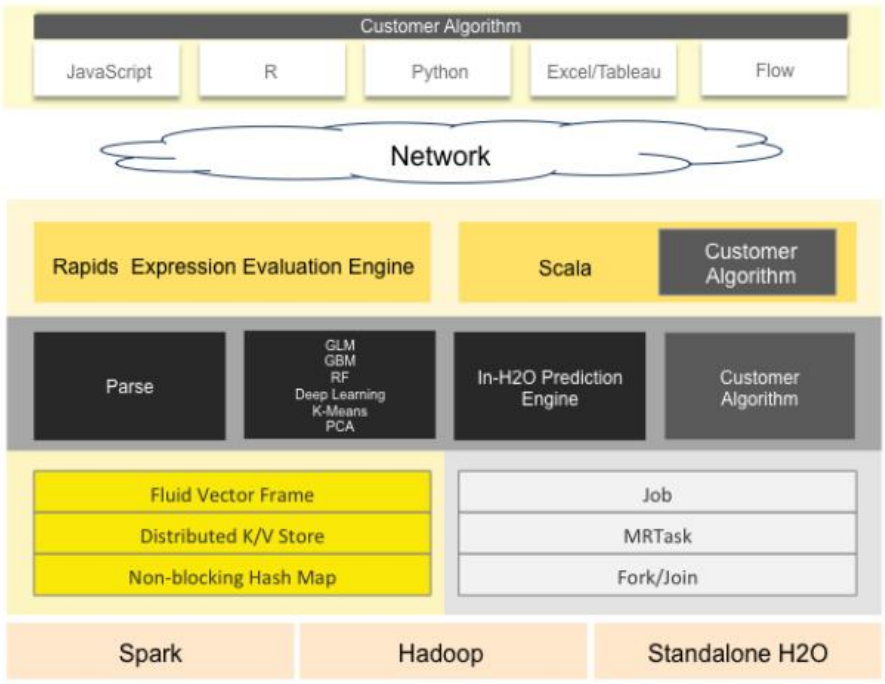
p-ISSN : 2808-8751 e-ISSN : 2798-2513



Gambar 1 teknologi H2O.AI. (Ellen, (2020)

Platform ML H2O-3

"Sederhananya, H2O-3 sangat terukur, terdistribusi, dalam memori, teknologi perangkat lunak sumber terbuka yang sangat cepat untuk membangun sistem AI dan ML (ML). Ellen, (2020) described "H2O-3 adalah mesin komputasi yang ditulis dan dijalankan sebagai aplikasi Java. Itu dapat dijalankan di tempat atau di cloud. H2O-3 memiliki antarmuka ke Python, R, JavaScript, dan beberapa lainnya. Pengguna menulis kode dalam bahasa pilihan mereka, antarmuka mendelegasikan pembuatan model ke mesin komputasi dan model yang diinginkan dikembalikan. Arsitekturnya ditunjukkan pada Gambar 3, dan contoh H2O-3 Python API ada di Listing 4.1, di mana data diimpor dari file CSV dan modelnya dilatih (H2O.AI. H2O Architecture (2020)



Gambar 2Arsitektur H2O-3 (H2O.AI. H2O Architecture (2020)

File input di Listing 4.1 diimpor dari sistem file Hadoop dan bisa berukuran besar. Platform perlu menyimpannya di memori cluster untuk komputasi. Biasanya tabelnya adalah array 2-D, tetapi dalam kasus ini, Java tidak mengizinkan array big data. Untuk mengatasi hal ini Distributed Key-Value Store (DKV) digunakan di dalam struktur data H2O-3 yang disebut Frame (H2O.AI. H2O Architecture (2020). Secara praktis, Frame adalah sebuah meja; artinya baris dan kolom. Dalam terminologi H2O-3, kolom adalah

p-ISSN : 2808-8751 e-ISSN : 2798-2513

Vektor, dan baris adalah item dari Vektor. Frame dapat diteruskan ke tugas Map/Reduce. Dalam kasus seperti itu, Frame yang diberikan dipisahkan menjadi Chunks (array of chunks). Ukuran array Chunk sama dengan jumlah Vektor. Di Potongan, bagian dari baris didistribusikan secara merata di sekitar cluster. Pada fase ini algoritma dapat mengubah data. Platform H2O-3 juga menyediakan API untuk kerangka kerja Map/Reduce, kerangka kerja Java yang memungkinkan program berjalan di cluster terdistribusi. Ini dirancang untuk menangani masalah koneksi dan perangkat keras tanpa mempengaruhi penyelesaian perhitungan. Waktu komputasi meningkat, tetapi dibandingkan dengan OpenMP/MPI tradisional, masalah ditangani secara otomatis tanpa kerjasama pemrogram. Intinya inti dari kerangka Map/Reduce terdiri dari dua operasi yakni Map dan Reudce.

Kode IF Asli

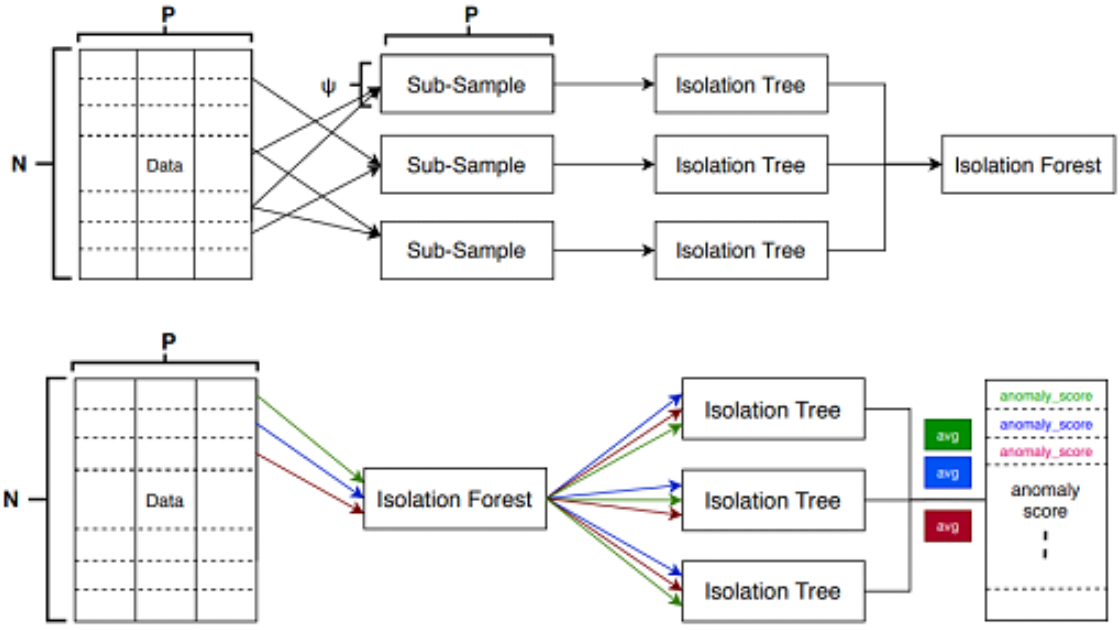
IF sudah diimplementasikan di perpustakaan H2O-3 (H2O.AI. IF, (2020). Karena algoritme EIF hanya menyesuaikan percabangan dan evaluasi IF, secara teoritis dimungkinkan untuk menyesuaikan implementasi IF yang ada. Dengan cara ini API yang ada dan uji kinerja dapat diambil alih sepenuhnya. EIF meningkatkan skor anomali dan kertas (Hariri et al., (2019) menggunakan peta warna untuk memvisualisasikannya. Oleh karena itu, peta warna yang sama dibuat. Implementasi Scikit-learn IF dan implementasi paper EIF dengan extensionLevel = 0 divisualisasikan bersamaan dengan implementasi H2O IF. Notebook Jupiter dengan hasilnya dapat ditemukan di CD terlampir. Inputnya adalah dua gumpalan 2-D Gaussian N ((0, 0), (1, 1)) dan N (10, 0),(1, 1). Inti dari H2O-3 IF adalah implementasi terdistribusi dari BST yang diimplementasikan untuk algoritma Random Forest, yang tidak bekerja dengan jumlah baris pada daun. Mempertimbangkan kinerja deteksi IF saat ini, fakta bahwa versi diperpanjang kehilangan interpretasi, dan yang terakhir, kemampuan, waktu, dan pengetahuan platform penulis, diputuskan bahwa IF Diperpanjang akan diimplementasikan sebagai model baru ke platform H2O 3.

Kemungkinan paralelisasi

Kemungkinan paralelisasi untuk kedua algoritma dan kedua tahap adalah sama. Bagian ini mendefinisikan kemampuan terkini dari IF yang diimplementasikan dan membahas potensi algoritme. Kemampuan memproses big data tidak dapat terpengaruh dalam skenario apa pun karena struktur data H2O-3 digunakan.

Dalam tahap pelatihan, data digunakan dalam mode read-only. Masalah kondisi balapan dihindari. IF membangun pohon independen dengan sub-sampel data acak. Setiap pohon dapat dibangun secara terpisah dengan mengacu pada sejumlah kecil data. Gedung BST juga bisa diparalelkan. Implementasi IF saat ini menggunakan struktur yang disebut DTree yang dikembangkan untuk algoritma Random Forest terdistribusi. Struktur DTree bertanggung jawab atas kinerja komputasi dalam implementasi IF saat ini.

p-ISSN : 2808-8751 e-ISSN : 2798-2513



Gambar 3 Diagram Tahap Pelatihan (atas), dan Tahap evaluasi (bawah)

Dalam tahap evaluasi, input untuk Hutan adalah deretan data. Baris h(x) dihitung pada setiap pohon dan rata-rata h(x) adalah input untuk rumus skor anomali. Panjang jalur tahapan evalusai di setiap pohon dapat dihitung secara independen. Map/Reduce dapat digunakan untuk ini. Pohon Isolasi dapat menghitung h(x) dalam metode Map, dan metode Reduce menambahkan h(x) pada baris keluaran yang benar. Tugas Map/Reduce kedua menghitung skor rata-rata dan anomali untuk setiap baris dalam output secara bersamaan. Implementasi IF saat ini hanya mendistribusikan data input dan penghitungan panjang jalur di setiap pohon. Ansambel pohon diulang secara berurutan. Pendekatan rekayasa perangkat lunak Simple and Clean Code vs. Performa, juga disebut Code clean, performa setelah simon, (2020); akmad, (2020); Mertz, (2020) akan diikuti selama implementasi EIF. Kode yang bersih dan dapat dibaca akan diimplementasikan terlebih dahulu, diikuti oleh implementasi semua API yang diperlukan untuk model dan yang terakhir, model akan diuji untuk deteksi anomali dan kinerja komputasi. Karena alat H2O-3 dibuat untuk skalabilitas tinggi, paralelisasi lebih lanjut harus diterapkan. Skalabilitas untuk input/output big data dimungkinkan selama struktur data yang benar digunakan. Jika panjang array bergantung pada jumlah baris, maka tugas Map/Reduce harus diimplementasikan. Jika panjangnya bergantung pada sejumlah fitur, maka operasi larik sederhana digunakan. Namun demikian, implementasi pertama akan didasarkan pada struktur Pohon Isolasi berurutan dan tahap evaluasi berurutan.

KESIMPULAN DAN SARAN

Implementasi Model

Tugas mengimplementasikan model ke dalam platform H2O-3 yang ada tidak berbeda dengan tugas pengembang perangkat lunak biasa lainnya. Status implementasi EIF dilacak dalam tugas penerbitan tiket PUBDEV-7138 (Maurerova, (2020). Tugas tersebut mencakup aktivitas standar, seperti memahami basis kode, memahami antarmuka, mengikuti konvensi kode yang diberikan, proses penggabungan fitur, dan banyak lagi. Menurut log pekerjaan, implementasinya memakan waktu sekitar 200MH - bagian singkat di bawah ini menjelaskan aktivitas utama yang telah dilakukan. Model EIF memiliki hyperparameter **ntrees** sebagai jumlah pohon dalam ansambel, **sample_size** sebagai ukuran sub-sampel dan **extension_level** dsebagai tingkat ekstensi. Kelas Java/Python yang diimplementasikan dan file R tercantum di bawah ini bersama dengan deskripsi singkat. Semua file yang diimplementasikan disediakan dalam CD tertutup dan juga tersedia dalam Pull Request (Valenta, (2020). ExtendedIsolationForest.java berisi tahapan pelatihan EIF, ExtendedIsolationForestTest.java berisi tes otomatis untuk EIF, termasuk tes dasar untuk tahap pelatihan serta metode pembantu. Sedangkan untuk ExtendedIsolationForestModel.java berisi tahapan

p-ISSN : 2808-8751 e-ISSN : 2798-2513

evaluasi EIF dan memegang pohon yang dilatih. ExtendedIsolationForestMojoWriter.java adalah kelas yang digunakan untuk menyimpan model di cluster. FilterGteTask.java dan FilterLtTask.java untuk memetakan/Mengurangi tugas untuk memfilter data dengan diberikan ~n dan ~p. Lihat Algoritma 5 baris 7 dan 8 untuk informasi lebih lanjut. SubSampleTask.java untuk memetakan/Mengurangi tugas untuk sub-sampling dari data yang diberikan. ExtendedIsolationForest.IsolationTree adalah seluruh implementasi Algoritma 5. Rekursi diganti dengan implementasi array dari BST, yang lebih cocok untuk digunakan dalam komputasi terdistribusi. gen_{alogrithm}.py (misalnya gen_python.py dan gen_R.py) merupakan kemampuan refleksi digunakan untuk menghasilkan API untuk berbagai bahasa pemrograman. File individual adalah kode pembuatan API untuk setiap bahasa. Sedangkan extendedisolationforest.R dan extended_isolation_forest.py adalah file Python dan R yang dibuat secara otomatis untuk algoritma EIF.

Pengujian Model

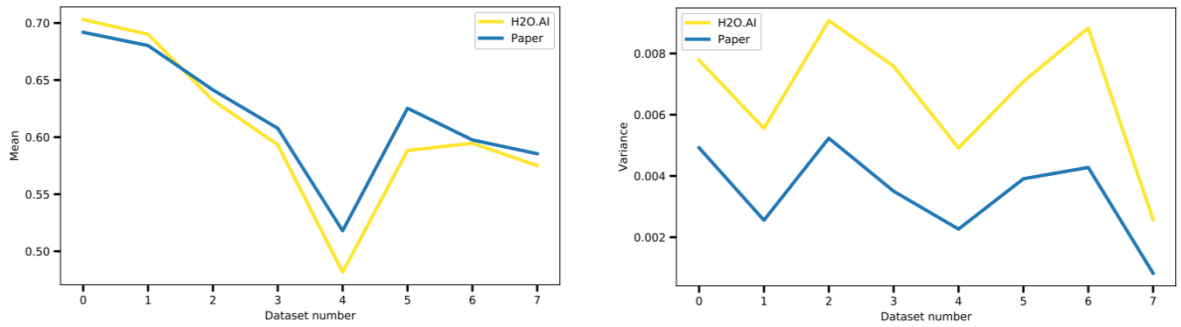
Pengujian dapat dibagi menjadi tiga proses. Pertama adalah pengujian Unit dari kelas yang diimplementasikan dan metode pembantu. Pengujian unit telah dilakukan karena merupakan kebijakan proyek untuk menyediakan pengujian otomatis untuk kode baru. Kedua, pengujian kinerja deteksi anomali pada data mainan. Ketiga adalah pengujian skalabilitas algoritma.

Kinerja deteksi anomali

Performanya diuji pada set data 2-D. Hariri et al., (2019) mengevaluasi kinerja deteksi anomali dengan membandingkan rata-rata dan varians dari skor anomali yang diberikan. Delapan set data dihasilkan, dengan masing-masing set data selanjutnya dilatih pada sepuluh model EIF H2O dan satu model dengan implementasi kertas EIF (karena waktu komputasi). Hasilnya ada pada Gambar 4.10. Rata-rata kedua algoritme serupa, tetapi varians untuk H2O EIF sekitar 0,004 lebih tinggi daripada implementasi kertas. Menurut hasil dari penelitian Hariri et al., (2019) itu berarti bahwa implementasi kertas memberikan deteksi yang lebih akurat dari struktur data yang diberikan. Ada kemungkinan bahwa metode tersebut mengalami bias dan lebih menyukai titik yang dipilih sebagai yang pertama dalam uji penjumlahan. CD terlampir berisi notebook Jupyter dengan kumpulan data teruji lainnya.

Skalabilitas

Skalabilitas tahapan pelatihan dan evaluasi diuji secara terpisah. Performa kemampuan skal dibandingkan dengan implementasi IF H2O-3.



Gambar 4 Kinerja deteksi anomali

Tahap pelatihan

Karena paralelisasi hanya relevan untuk input big data, skalabilitas bergantung pada parameter ukuran sampel. Untuk nilai kecil, mis. 256 yang direkomendasikan, komputasi berurutan kecuali untuk sub-sampling. Untuk nilai ukuran sampel di mana Map/Reduce membantu, performa sangat bergantung pada jumlah utas. Untuk ukuran sampel yang besar, DTree berfungsi dengan baik dan layak menerapkan BST sebagai versi terdistribusi. Jika ukuran sampel kecil maka DTree memperlambat komputasi. Pada g=gambar 5 IF yang Diperluas bekerja lebih baik pada big data ketika ukuran sampel dijaga tetap kecil. Dalam hal ini waktu komputasi lebih rendah bahkan dengan lebih sedikit utas. IF tampil lebih buruk dengan ukuran sampel yang kecil kemungkinan besar karena struktur DTree. Di sisi lain, IF berkinerja lebih baik secara signifikan dengan ukuran sampel yang besar. Jumlah node iTree di EIF tergantung pada ukuran sampel yang ditentukan, dalam IF fungsi yang sama disediakan oleh atribut max depth. Forest berperforma

p-ISSN : 2808-8751 e-ISSN : 2798-2513

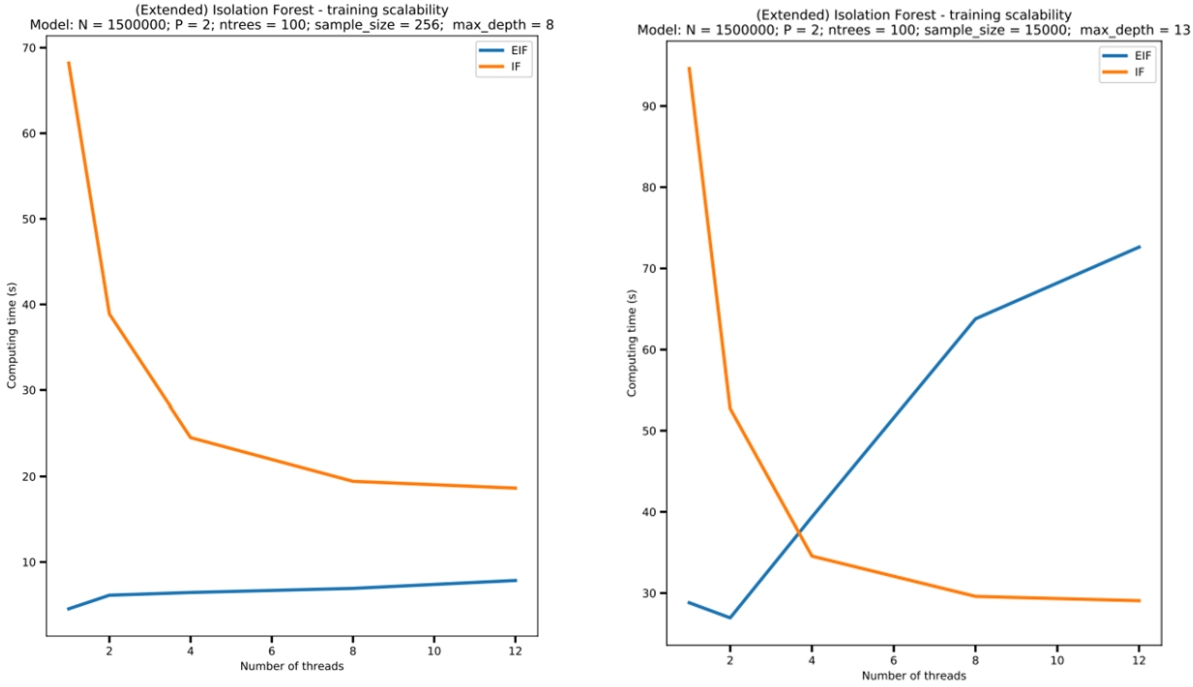
lebih baik pada big data saat ukuran sampel tetap kecil. Dalam hal ini waktu komputasi lebih rendah bahkan dengan lebih sedikit utas. IF tampil lebih buruk dengan ukuran sampel yang kecil kemungkinan besar karena struktur DTree. Di sisi lain, IF berkinerja lebih baik secara signifikan dengan ukuran sampel yang besar. Jumlah node iTree di EIF tergantung pada ukuran sampel yang ditentukan, dalam IF fungsi yang sama disediakan oleh atribut max depth.

Tahap evaluasi

Tahap evaluasi melakukan hal yang sama untuk kedua implementasi. Ketergantungan logarithmic pada jumlah thread terlihat jelas pada Gambar 6. Dalam hal ini tidak masalah jika iTree didistribusikan, juga tidak masalah berapa nilai ukuran sampel.

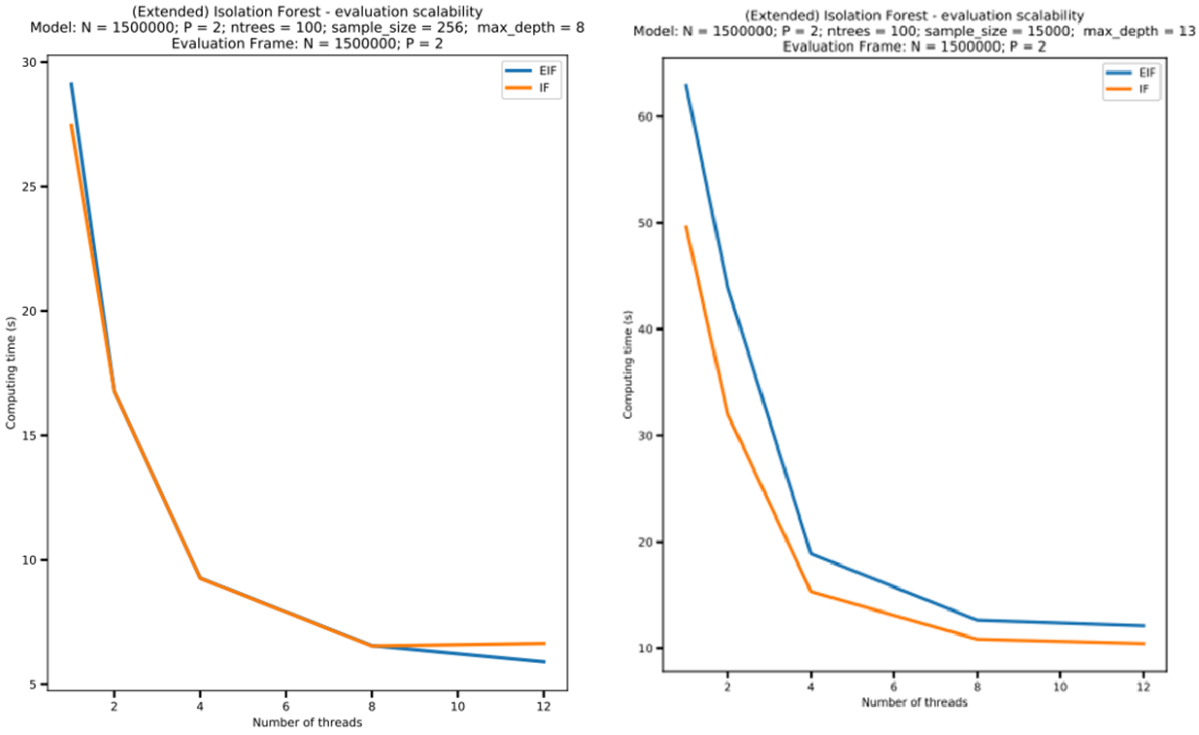
Hasil Uji Skalabilitas

Tahap evaluasi memiliki kinerja yang baik untuk kedua algoritma tersebut. Waktu komputasi lebih kecil logaritmik dengan jumlah utas. Ada pilihan untuk menjadi lebih terukur tetapi dari sudut pandang rekayasa perangkat lunak, tidak ada nilai tambah yang signifikan yang akan membenarkan upaya tersebut. Nilai tambah terletak pada tahap pelatihan. Karena ukuran sampelnya kecil, algoritma sekuensial bekerja lebih baik. Ketika ukuran sampel bertambah, waktu komputasi IF secara logaritmik lebih kecil dengan jumlah thread, tetapi waktu komputasi EIF bertambah dengan jumlah thread karena komunikasi antar thread tanpa nilai tambah dalam waktu komputasi. Untuk memperbaiki masalah EIF ini, kedua opsi skalabilitas sama-sama dapat diterima oleh pengguna model. Jika datanya besar dan ukuran sampel tetap kecil, maka jumlah pohon kemungkinan besar akan bertambah, dan dalam hal ini, setiap pohon dapat dibangun secara independen dengan algoritma sekuensial. Jika ukuran sampel meningkat, maka jumlah pohon kemungkinan besar tetap kecil dan BST terdistribusi lebih membantu daripada algoritma sekuensial.



Gambar 5 Skalabilitas - tahap pelatihan dengan ukuran sampel =256 dan 15.000

p-ISSN : 2808-8751 e-ISSN : 2798-2513



Gambar 6 Skalabilitas - tahap evaluasi dengan ukuran sampel=256 dan 15.000

KESIMPULAN DAN PENELITIAN DIMASA DEPAN

Penelitian ini membahas implementasi algoritma EIF ke dalam platform sumber terbuka ML H2O-3. Algoritma deteksi Anomali diperkenalkan - baik dari bidang ML (K-means, DBSCAN, SVM) maupun dari cabang DL (Robust Deep Autoencoder). Selanjutnya gagasan tentang IF, dasar pemikiran dari IF yang Diperluas direprenstasikan dan dilanjutkan dengan implementasi IF yang Diperpanjang. Implementasi berhasil dan status saat ini adalah Permintaan Tarik menunggu peninjauan. Hasil pengujian menunjukkan bahwa Extended Isolation Model perlu disesuaikan. Tes kinerja deteksi anomali mengungkapkan sedikit ketidaksempurnaan dalam deteksi struktur data jika dibandingkan dengan satu-satunya implementasi algoritma Python yang tersedia. Masalah ini dapat diperbaiki dengan sub-sampling yang lebih baik atau di percabangan iTree, di mana algoritme bisa menghindari titik perpecahan dengan leluhur kosong. Tes skalabilitas dilakukan dan hasilnya dibandingkan dengan implementasi IF H2O-3 saat ini. Ujian untuk tahap evaluasi dinyatakan lulus. Waktu komputasi secara logaritmik lebih kecil dengan jumlah utas. Di sisi lain, kesenjangan kinerja ditemukan pada tahap Pelatihan. Untuk IF yang Diperluas, waktu komputasi tidak berubah ketika nilai ukuran sampel dijaga tetap kecil. Namun, dengan ukuran sampel yang lebih besar, persyaratan waktu komputasi untuk EIF meningkat, sedangkan waktu komputasi untuk IF menjadi lebih kecil secara logaritma dengan jumlah thread di semua kasus. Kinerja pelatihan dapat ditingkatkan dengan membangun pohon secara paralel. Bahkan bisa mengungguli IF saat ini. Kemungkinan lain untuk meningkatkan kinerja adalah dengan melakukan proses percabangan pohon secara paralel. Saya akan terus mengembangkan implementasi EIF bekerjasama erat dengan H2O - tidak hanya untuk menyelesaikan masalah tersebut tetapi untuk lebih meningkatkannya. Status penyebaran siap produksi ada di tiket sistem pelacakan masalah. Seperti disebutkan dalam makalah EIF, algoritme dapat ditingkatkan lebih lanjut dengan menskalakan anomali deteksi untuk data dimensi tinggi. Ini dapat diimplementasikan dengan menambahkan parameter lain yang memungkinkan metode pemilihan fitur dalam perhitungan.

REFERENCES

p-ISSN : 2808-8751 e-ISSN : 2798-2513

1. Ahmed, M.; Mahmood, A. N.; et al. A survey of network anomaly detection techniques. Journal of Network and Computer Applications, volume 60, 2016.
2. akmad. Performance vs Readability. Stack Overflow, [cit. 2020-04-17]. Available from: https://stackoverflow.com/questions/30754/performance-vs-readability
3. Caffe. Development and Contributing. Caffe, [cit. 2020-04-17]. Available from: https://caffe.berkeleyvision.org/development.html
4. Chalapathy, R.; Chawla, S. Deep Learning for Anomaly Detection: A Survey. CoRR, volume abs/1901.03407, 2019, 1901.03407. Available from: http://arxiv.org/abs/1901.03407
5. Chandola, V.; Banerjee, A.; et al. Anomaly Detection: A Survey. ACM Comput. Surv., volume 41, 07 2009, doi:10.1145/1541880.1541882.
6. DataScience.com. Introduction to Anomaly Detection. KDnuggets, [cit. 2020-04-17]. Available from: https://www.kdnuggets.com/2017/04/ datascience-introduction-anomaly-detection.html
7. Ding, Z.; Fei, M. An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data using Sliding Window. IFAC Proceedings Volumes, volume 46, no. 20, 2013: pp. 12 – 17, ISSN 1474- 6670, doi:https://doi.org/10.3182/20130902-3-CN-3020.00044, 3rd IFAC Conference on Intelligent Control and Automation Science ICONS 2013. Available from: http://www.sciencedirect.com/science/article/pii/S1474667016314999
8. Ellen Friedman, P. AI & ML Platforms: My Fresh Look at H2O.ai Technology. H2O.AI, [cit. 2020-04-17]. Available from: https://www.h2o.ai/blog/ai-ml-platforms-my-fresh-look-at-h2o-ai-technology/
9. Ester, M.; Kriegel, H.-P.; et al. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, 1996, pp. 226–231.
10. Foundation, T. A. S. Guide to uploading artifacts to the Central Repository. The Apache Software Foundation, [cit. 2020-04-17]. Available from: https://maven.apache.org/repository/guide-centralrepository-upload.html
11. GitHub. GitHub. GitHub, [cit. 2020-04-17]. Available from: https://github.com/
12. Gumbao, M. G. Best clustering algorithms for anomaly detection. Towards Data Science, [cit. 2020-04-17]. Available from: https://towardsdatascience.com/best-clustering-algorithmsfor-anomaly-detection-d5b7412537c8
13. H2O.AI. AutoML: Automatic Machine Learning. H2O.AI, [cit. 2020-04-17]. Available from: http://docs.h2o.ai/h2o/latest-stable/h2odocs/automl.html
14. H2O.AI. H2O Architecture. H2O.AI, [cit. 2020-04-17]. Available from: https://docs.h2o.ai/h2o/latest-stable/h2o-docs/architecture.html
15. H2O.AI. H2O.ai is Democratizing Artificial Intelligence. H2O.AI, [cit.2020-04-17]. Available from: https://www.h2o.ai/company/
16. H2O.AI. H2O.ai Products and Solutions. H2O.AI, [cit. 2020-04-17]. Available from: https://www.h2o.ai/products/
17. H2O.AI. Isolation Forest. H2O.AI, [cit. 2020-05-17]. Available from: https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/if.html
18. Hariri, S.; Carrasco Kind, M.; et al. Extended Isolation Forest. IEEE Transactions on Knowledge and Data Engineering, 2019: p. 1–1, ISSN 2326-3865, doi:10.1109/tkde.2019.2947676. Available from: http://dx.doi.org/10.1109/TKDE.2019.2947676
19. Hariri, S.; Carrasco Kind, M.; et al. Extended Isolation Forest. IEEE Transactions on Knowledge and Data Engineering, 2019: pp. 1–1, doi:10.1109/TKDE.2019.2947676. Available from: https://github.com/sahandha/eif
20. joelbarmettlerUZH. How to upload your python package to PyPi. Medium, [cit. 2020-04-17]. Available from: https://medium.com/@joel.barmettler/how-to-upload-your-python-package-to-pypi65edc5fe9c56

p-ISSN : 2808-8751 e-ISSN : 2798-2513

21. Li, X.; Lu, Y.; et al. Out-of-Distribution Detection for Skin Lesion Images with Deep Isolation Forest. 2020, 2003.09365.

22. Liu, F. T.; Ting, K.; et al. Isolation Forest. 01 2009, doi:10.1109/ ICDM.2008.17.

23. Lutins, E. DBSCAN: What is it? When to Use it? How to use it. Medium, [cit. 2020-04-17]. Available from: https://medium.com/@elutins/dbscan-what-is-it-when-to-use-ithow-to-use-it-8bd506293818

24. MAHOUT. How to contribute. MAHOUT, [cit. 2020-04-17]. Available from: https://mahout.apache.org/developers/how-to-contribute

25. Matousek, J. AutoML - NAHRAD´I ROBOTI ANALYTIKY? Data Mind, [cit. 2020-04-21]. Available from: http://docs.h2o.ai/h2o/lateststable/h2o-docs/automl.html

26. Maurerova, V. Implement Extended Isolation Forest. H2O.AI, [cit. 2020-05-25]. Available from: https://0xdata.atlassian.net/browse/PUBDEV-7138

27. Mertz, A. Simple and Clean Code vs. Performance. Simplify C++!, [cit. 2020-04-17]. Available from: https://docs.h2o.ai/h2o/lateststable/h2o-docs/architecture.html

28. Mishra, A. Swamping and Masking in Anomaly detection: How Subsampling in Isolation Forests helps mitigate this? Medium, [cit. 2020-04-17]. Available from: https://medium.com/walmartlabs/swamping-andmasking-in-anomaly-detection-how-subsampling-in-isolationforests-helps-mitigate-bb192a8f8dd5

29. Munz, G.; Li, S.; et al. Traffic anomaly detection using k-means clustering. In GI/ITG Workshop MMBnet, 2007, pp. 13–14.

30. PoradaKev. Error while installing eif. GitHub, [cit. 2020-04-17]. Available from: https://github.com/sahandha/eif/issues/14

31. Roy, S. 13 Open-Source Artificial Intelligence and Machine Learning Tools to Watch in 2019. DEV, [cit. 2020-04-17]. Available from: https://dev.to/promozseo/13-open-source-artificial-intelligenceand-machine-learning-tools-to-watch-in-2019-1hmc

32. Schubert, E.; Sander, J.; et al. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. ACM Trans. Database Syst., volume 42, no. 3, July 2017, ISSN 0362-5915, doi:10.1145/3068335. Available from: https://doi.org/10.1145/3068335

33. Scikit-learn. Contributing. Scikit-learn, [cit. 2020-04-17]. Available from: https://scikit-learn.org/stable/developers/contributing.html

34. Scikit-learn. IsolationForest. Scikit-learn, [cit. 2020-04-17]. Available from: https://scikit-learn.org/stable/modules/generated/ sklearn.ensemble.IsolationForest.html

35. simon. Should a developer aim for readability or performance first? [closed]. Stack Overflow, [cit. 2020-04-17]. Available from: https://stackoverflow.com/questions/183201/should-adeveloper-aim-for-readability-or-performance-first

36. TensorFlow. Contribute to TensorFlow. TensorFlow, [cit. 2020-04-17]. Available from: https://www.tensorflow.org/community/contribute

37. Valenta, A. PUBDEV-7138 Extended Isolation Forest. H2O.AI, [cit. 2020-05-25]. Available from: https://github.com/h2oai/h2o-3/pull/4319

38. Wikipedia contributors. Euler–Mascheroni constant — Wikipedia, The Free Encyclopedia. 2020, [Online; accessed 8-May-2020]. Available from: https://en.wikipedia.org/w/index.php?title=Euler%E2%80%93Mascheroni_constant&oldid=954141487

39. Zhang, X.; Gu, C.; et al. Support Vector Machines for Anomaly Detection. 01 2006, pp. 2594 – 2598, doi:10.1109/WCICA.2006.1712831.

40. Zhou, C.; Paffenroth, R. C. Anomaly Detection with Robust Deep Autoencoders. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, New York, NY, USA: Association for Computing Machinery, 2017, ISBN 9781450348874, p. 665–674, doi:10.1145/3097983.3098052. Available from: https://doi.org/10.1145/3097983.3098052

p-ISSN : 2808-8751 e-ISSN : 2798-2513

41. Zimek, A.; Filzmoser, P. There and back again: Outlier detection between statistical reasoning and data mining algorithms. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, volume 8, no. 6, Nov. 2018, ISSN 1942-4787, doi:10.1002/widm.1280.